Rx: Treating bugs as allergies

Motivation
- Software failures greatly reduce system availability
  - Software defects cause 40% of system failures
  - Memory and concurrency bugs cause over 60% of system vulnerabilities
  - Software bugs still slip through even the strictest testing
- Many bugs are induced by "allergens" - characteristics of the execution environment

Advantages of Rx
- Comprehensive: survives not only nondeterministic bugs, but also deterministic bugs
- Safe: does not speculatively “fix” bugs
- Noninvasive: requires few to no modification to applications’ source code
- Efficient: requires no rebooting or warming up
- Informative: provides additional diagnostic information for postmortem analysis

Limitations of Previous Approaches
- Restarting techniques:
  - It is suited for surviving software failures
  - Service is unavailable while restarting, which takes up to several seconds
- General checkpointing and recovery:
  - Cannot deal with deterministic bugs
- Application-specific recovery:
  - Multi-process model cannot restore corrupted shared data to a consistent state
  - Others require software to be failure-aware
- Failure-oblivious computing and the reactive immune system:
  - Unsafe to use for correctness-critical applications
- Existing work provides insufficient feedback to developers for debugging

Main Idea
- Real life analogies:
  - Allergic to milk → Install an air filter to remove pollen from the air
  - Allergic to milk → Remove dairy products from the diet
- Could be unhealthy
- Treating bugs as allergies:
  - Checkpoint programs periodically
  - If the program fails, rollback to previous state, modify the environment, and re-execute

Environmental Changes
- Environmental changes are made during program re-execution
- Possible environmental changes and their potential-avoided bugs:

<table>
<thead>
<tr>
<th>Category</th>
<th>Environmental Changes</th>
<th>Potentially-Avoided Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Management</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dangling reference</td>
<td>Double free, dangling pointer</td>
</tr>
<tr>
<td></td>
<td>Lost orphaned memory</td>
<td>Dynamic buffer overflows</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memory corruption</td>
</tr>
<tr>
<td>Scheduling</td>
<td></td>
<td>Message Reordering</td>
</tr>
<tr>
<td>Concurrency Bugs</td>
<td></td>
<td>Uninterrupted read</td>
</tr>
<tr>
<td>User-Related Bugs</td>
<td></td>
<td>Concurrency Bugs</td>
</tr>
</tbody>
</table>

Rx Design
- Sensors:
  - Detect and identify software failures at run time
- Checkpoint-and-rollback component:
  - Takes checkpoints of the server and rolls it back to some previous checkpoint
- Environment wrappers:
  - Change execution environment during re-execution
- Proxy:
  - Makes server recovery process transparent to clients
- Control unit:
  - Maintains checkpoints and devises recovery strategy

Results (I)
- Experiments setup:
  - Two machines: one for server, one for clients
  - Four real-world server applications
  - MySQL: database server
  - Apache: web server
  - CVS: source code control server
- Overall result:
  - Survive 6 bugs in 4 server applications
  - Recovery time of .017 to .126 seconds

Results (II)
- Rx vs. restart:
  - Rx provides similar performance as baseline
  - Rx hides server failures, while restart cannot